

Tree-like resolution complexity of two planar problems

Dmitry Itsykson^{*1}, Anna Malova^{†2}, Vsevolod Oparin^{‡2}, and Dmitry Sokolov^{§1}

¹St. Petersburg Department of V.A. Steklov Institute of Mathematics of the Russian Academy of Sciences

²St. Petersburg Academic University of the Russian Academy of Sciences

December 4, 2014

Abstract

We consider two CSP problems: the first CSP encodes 2D Sperner's lemma for the standard triangulation of the right triangle on n^2 small triangles; the second CSP encodes the fact that it is impossible to match cells of $n \times n$ square to arrows (two horizontal, two vertical and four diagonal) such that arrows in two cells with a common edge differ by at most 45° , and all arrows on the boundary of the square do not look outside (this fact is a corollary of the Brouwer's fixed point theorem). We prove that the tree-like resolution complexities of these CSPs are $2^{\Theta(n)}$. For Sperner's lemma our result implies $\Omega(n)$ lower bound on the number of request to colors of vertices that is enough to make in order to find a trichromatic triangle; this lower bound was originally proved by Crescenzi and Silvestri.

CSP based on Sperner's lemma is related with the PPAD-complete problem. We show that CSP corresponding to arrows is also related with a PPAD-complete problem.

1 Introduction

The resolution proof system is one of the simplest and well-studied proof systems. There are well known methods of proving lower and upper bounds on the complexity of several types of formulas. However there are no known universal methods that may be used to determine the asymptotic resolution complexity of a given family of formulas.

^{*}dmitrits@pdmi.ras.ru, partially supported by the RFBR grant 14-01-00545, by the President's grant MK-2813.2014.1, by the Government of the Russia (grant 14.Z50.31.0030) and by the RAS program of fundamental research

[†]malova.any@gmail.com

[‡]oparin.vsevolod@gmail.com, partially supported by ANR NAFIT 008-01

[§]sokolov.dmt@gmail.com, partially supported by the RFBR grant 12-01-31239-mol-a, by the President's grant MK-2813.2014.1, by the Government of the Russia (grant 14.Z50.31.0030)

Baker [1] extended resolution proof system for constraint satisfaction problems (CSP) under arbitrary alphabets. The resolution proof system is connected with backtracking algorithms (so called DPLL algorithms). Every DPLL algorithm for CSP under k -element alphabet creates k -ary tree. Every node of this tree corresponds to a variable, and edges, going to children, correspond to k different substitutions of the variable. Every leaf of the tree contains a constraint that is falsified by substitution made on the path from the root to that leaf. We call such tree as contradiction search tree. It is well known that the minimal size of a contradiction search tree for an unsatisfiable CSP is equal to the size of the minimal tree-like resolution proof of CSP.

Every unsatisfiable constraint satisfaction problem F under k -element alphabet has three parameters: $S(F)$ is the minimal size of resolution proof of F , $S_T(F)$ is the minimal size of tree-like resolution proof, and $d(F)$ is the minimal depth of contradiction search tree. These parameters are connected by trivial inequalities: $S(F) \leq S_T(F) \leq k^{d(F)}$. The paper [2] presents the family of formulas that exponentially separate S and S_T , and paper [8] gives an example of family F_n such that $S_T(F_n) = O(n)$ and $d(F_n) = \Omega(n/\log n)$. The number $d(F)$ is the worst-case lower bound on the number of requests to the variables of CSP that is necessary to do in order to find a falsified constraint.

We consider the constraint satisfaction problem that codes 2D Sperner lemma: namely is impossible to color vertices of the standard triangulation of the right triangle in three colors such that vertices of the right triangle are colored in different colors and all vertices on the side are colored in two colors and there are no small triangles cored with three different colors. We prove that that size of tree-like refutation of this CSP is at least $2^{\Omega(n)}$, where n is the number of points on the side of the triangle. Previous result by Crescenzi and R. Silvestri [4] gives a linear lower bound on the depth of resolution proof for the same CSP.

We also consider the constraint satisfaction problem that codes that it is impossible to match cells of the $n \times n$ square with arrows from the set $\{\leftarrow, \nearrow, \uparrow, \searrow, \rightarrow, \swarrow, \downarrow, \nwarrow\}$ such that arrows in every two adjacent cells differ by at most 45° and boundary arrows are not directed outside the square. The impossibility is a corollary of the Brouwer's fixed point theorem. We prove that tree-like resolution complexity of this CSP is equal to $2^{\Theta(n)}$.

Our research is motivated by the investigation of complexity classes PPA, PPAD, PPADS, PPP that are subclasses of TFNP function problems that are guaranteed to have a solution because of unsatisfiability of certain CSP [6]. It is known that Sperner's Lemma corresponds to PPAD-complete problem. We show that CSP with arrows also corresponds to PPAD-complete problem.

2 Preliminaries

Let $X = \{x_1, x_2, \dots, x_n\}$ be a finite number of variables that can take values from a finite alphabet $W = \{w_1, w_2, \dots, w_k\}$. Let S be the set of constraints for X : every constraint depends on some subset $X' \subseteq X$ and defines a set of values that variables from X' can take simultaneously. A constraint satisfaction problem (CSP) is a triplet $\langle X, W, S \rangle$. A constraint satisfaction problem is satisfiable if there is a set of values for X that satisfies all constraints from S , otherwise we call CSP unsatisfiable.

A *partial substitution* is a map from X to $W \cup \{*\}$. We say that substitution ρ sets value for variable x if $\rho(x) \neq *$. A substitution is *full* if values of all variables are set.

Substitution ρ falsifies constraint $C \in S$ if values of all variables, that constraint C depends on, are defined by ρ and constraint C forbids such a setting of values by ρ .

A *contradiction search tree* for an unsatisfiable CSP is a rooted k -arity tree such that vertices are marked with variables and edges, that connect a vertex marked with x to its children, are marked with substitutions $x := w$ for every $w \in W$. Every leaf of the tree is marked with a constraint that is refuted by the substitution made along the path from the root to the leaf.

A nogood is a constraint of the following form: $\neg(x_1 = a_1 \wedge \dots \wedge x_\ell = a_\ell)$, where $x_1, \dots, x_\ell \in X, a_1, \dots, a_\ell \in W$. In the case of binary alphabet $W = \{0, 1\}$ nogoods are equivalent to clauses. If constraint depends on ℓ variables, then it can be represented as a conjunction of at most $|W|^\ell$ nogoods.

The resolution proof system can be generalized to CSP if all its constraints are represented as conjunctions of nogoods. A resolution proof for some CSP formula ϕ is a sequence of nogoods C_1, C_2, \dots, C_t , that ends with empty nogood \square . Every nogood is either contained in ϕ or can be derived from k previous nogoods by the resolution rule. Let $\{N_a\}_{a \in W}$ be the set of nogoods in the following form: $N_a = \neg(x = a \wedge \alpha_a)$ for all $a \in W$. Then nogood $\neg(\bigwedge_{a \in W} \alpha_a)$ is a resolvent (a result of resolution rule) of nogoods $\{N_a\}_{a \in W}$. A resolution proof is called *tree-like* if there exists a k -ary tree such that its leaves are marked with nogoods of initial formula ϕ and nogood in any other vertex v is a resolvent of nogoods that are written in children of v ; the root of the tree is marked with the empty nogood \square .

Proposition 2.1 ([1][5]). The size of the minimal tree-like resolution proof for every unsatisfiable CSP formula ϕ is equal to the size of minimal contradiction search tree for ϕ .

3 Tree size lower bounds

Consider CSP ϕ under a finite alphabet $W = \{w_1, w_2, \dots, w_k\}$ that depends on variables $X = \{x_1, x_2, \dots, x_n\}$ and consists of constraints C_1, C_2, \dots, C_m .

We consider the following game that will be used for proving lower bounds on the size of contradiction search trees. A game is defined by an unsatisfiable CSP formula ϕ ; two players Alice and Bob play as follows. Alice secretly from Bob chooses a contradiction search tree for ϕ and put a token in the root of the tree. Alice asks Bob about the value of a variable that corresponds a vertex that contains the token. Bob either returns a value of asked variable or suggest to Alice choose the value by herself from some subset of W of cardinality at least 2. In the second case we say that Bob moves ChooseAny. Alice moves the token according the Bob's answer, if Bob moves ChooseAny, then Alice chooses a value herself from the set proposed by Bob. The game is over if the token is in a leaf, that is a contradiction is found. The goal of Bob is to maximize the number of ChooseAny answers along the path from the root to a leaf.

Lemma 3.1. Let for CSP formula ϕ there exist such a strategy of Bob that for all strategies of Alice, Bob moves ChooseAny at least t times. Then the size of any contradiction search tree for ϕ is at least 2^t .

Proof. Consider some contradiction search tree T for ϕ . We construct probabilistic distribution on the leaves of T that corresponds to Bob's strategy and the following randomized strategy of Alice. Alice chooses the tree T and if Bob moves ChooseAny, Alice chooses a value at random with equal probabilities. By the statement of the Lemma the probability that the game will finish in every particular leaf is at most 2^{-t} . Since with probability 1 the game will finish in a leaf, the number of leafs is at least 2^t . \square

In applications of Lemma 3.1 it is convenient to describe the strategy of Bob in the following terms: Bob has a partial substitution that assigns values to variables that Alice asked for and probably to some other variables. The current substitution should not falsify constraints. If Alice asks for a variable that is determined by the substitution, Bob returns the assigned value, otherwise Bob moves ChooseAny and extends the substitution by the value that Alice chooses and probably substitutes values to some other variables.

3.1 Sperner's lemma

Let us consider triangle T , each side of whom is divided on $n - 1$ segments of the same length. Denote the ends of the segments by $p_1, \dots, p_n, p_n = q_1, \dots, q_n, q_n = r_1, \dots, r_n = p_1$ as it is shown on Figure 1.

The standard triangulation is obtained by joining the following pairs of vertices:

1. p_i and $q_{n-(i-1)}, \forall i(2 \leq i \leq n - 1)$
2. p_i and $r_{n-(i-1)}, \forall i(2 \leq i \leq n - 1)$
3. q_i and $r_{n-(i-1)}, \forall i(2 \leq i \leq n - 1)$

The crossings between the above straight-line segments are called crossing vertices.

Denote all vertices in the triangle by S . A vertex coloring $c : S \rightarrow \{Red, Green, Blue\}$ is said to be Sperner's coloring if vertices p_1, q_1 , and r_1 are labeled by three different colors and each vertex on each edge of triangle T is colored by only one of two colors of the ends of this edge.

In the triangulation we will discuss only small triangles, i.e. triangles without crossings inside. A triangle in the triangulation is said to be *trichromatic* if all its vertices are colored with three different colors.

Lemma 3.2 (Sperner [7]). If c is a Sperner coloring, the triangulation contains a trichromatic triangle.

Let denote the unsatisfiable CSP corresponding to Lemma 3.2 by ϕ_n . Variables correspond to the vertices of the triangulation and can take values from alphabet $\{Red, Green, Blue\}$. Three constraints fix particular colors for vertices of triangle T and we set a constraint for each vertex on each side of T , which fixes a set of two possible colors. Also for each small triangle in the triangulation we set a constraint which forbids its trichromatic coloring. The unsatisfiability of ϕ_n immediately follows from Sperner's lemma.

Lemma 3.3. For CSP ϕ_n there exists such a strategy of Bob that gives $\Omega(n)$ ChooseAny answers for any strategy of Alice.

Before the proof we associate variables of the CSP with vertices of the triangulation and a partial substitution with a coloring. Some vertices can be non-colored, so values of the corresponding vertices has not been set.

Proof. We provide the strategy of Bob and then prove the lower bound.

As it is described in the previous section, Bob maintains a coloring. If Alice asks Bob for a colored vertex, Bob returns its color.

Bob maintains a path P which starts from $r_{\lceil \frac{n}{2} \rceil}$ and stops at a crossing vertex f . Path P goes along edges of the triangulation and can use only right or right-down directions. Bob keeps in mind two segments which start from vertex f and go in right and right-down directions to the border of the triangle. We denote them by RP and DP , respectively. We denote the parallelogram, surrounded by RP , DP and the border of triangle T , by H .

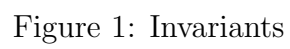
Buffer is an area that is strictly in parallelogram H and contains all points whose distance to RP or DP are at most two. Note that the buffer does not include points on RP and DP .

Informally Bob hides a trichromatic triangle in parallelogram H . Formally he maintains the following invariants (see Figure 1):

1. The border vertices are colored according the following
 - (a) $c(p_1) = c(r_i) = \text{Blue}$, for any i , with $\lceil \frac{n}{2} \rceil - 1 \leq i \leq n - 1$
 - (b) $c(q_n) = c(r_i) = \text{Green}$, for any i , with $2 \leq i \leq \lceil \frac{n}{2} \rceil$
 - (c) $c(p_i) = c(q_j) = \text{Red}$, for any i, j , with $2 \leq i \leq n, 2 \leq j \leq n - 1$
2. Vertices in the buffer are empty. It means Alice has not asked for them.
3. Any vertex on path P is green, and any vertex just below the path is blue.
4. Any vertex above $P \cup RP$ is colored, and each blue vertex is surrounded by red vertices.
5. Any vertex left to $P \cup DP$ is colored, and each green vertex is surrounded by red vertices.
6. Any vertex on RP is green, any vertex on DP is blue.
7. Any green or blue vertex in H , excluding RP and DP , is surrounded by red vertices.
8. All uncolored vertices are in H .

It is easy to see that if all invariants are hold, then there are no trichromatic triangle in the current coloring.

Absence of such a triangle above $P \cup RP$ (including border $P \cup RP$) follows from Invariant 4. There is not adjacent vertices colored on blue and green. Similarly absence of a trichromatic triangle left to $P \cup DP$ follows from Invariant 5. Invariant 3 guarantees that vertices on P can not lie on a trichromatic triangle, because all these vertices don't have adjacent red vertex below. Finally Invariants 2 and 7 implies that there are no trichromatic triangles in H .



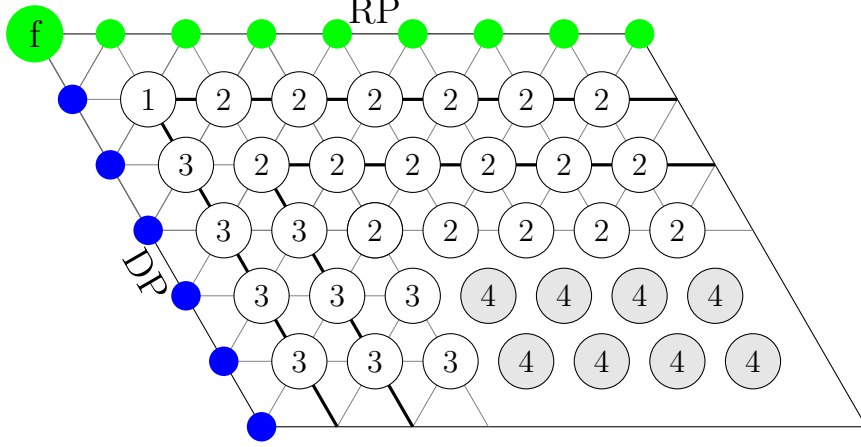


Figure 2: Cases

We will prove that if after a query of Alice, Bob cannot support these invariants, he has already given $\Omega(n)$ ChooseAny answers.

Initially $f = r_{\lceil \frac{n}{2} \rceil}$, P contains only f . Segments RP and DP are defined by the current position of vertex f . In the current coloring all vertices in area above RP are colored by green and ones left to DP by blue.

Let Alice ask for a non-colored vertex. By Invariant 8 such a vertex lies in parallelogram H . Bob answers accordingly the following cases, which are shown on Figure 2.

In Case 4 Bob returns ChooseAny(Green, Blue) and surrounds selected vertex by red ones to hold Invariant 7.

In cases 1 – 3 Bob has to move the buffer to hold Invariant 2. In order to explain how to move the buffer, we introduce two new notions: a slot and an expanded slot. The parallelogram H is a union of horizontal and diagonal segments. The crossings of these segments are vertices of the triangulation. For the horizontal segments we skip first three ones on the top and divide all others onto the groups with four segments in each. We do the same thing with the vertical segments skipping three leftmost ones. We ignore the last segments of each type if their number is at most 3.

An *expanded slot* is a union of two groups of horizontal and diagonal groups of segments.

For an expanded slot we consider the parallelogram which is formed by the borders of triangle T , the next to the leftmost segment in the vertical group and the next to top segment in horizontal one. A *slot* is a subset of the considered expanded slot which also lies in the described parallelogram.

The left and top borders of a slot can be used for new RP and DP , respectively. The rest of the slot can be used for the new buffer.

We say that a group of segments is clear if Alice has not requested any vertex in this group.

If Alice's request fits Cases 1 – 3, Bob looks for the first clear horizontal and diagonal groups of segments starting from vertex f , forms an expanded slot and extends path P as it will be described below. The new end of path P defines new position of parallelogram H and segments RP and DP . Bob adds all vertices, which lie outside of new parallelogram H or belong segment RP and DP , in the partial substitution as it is described in Invariants 4

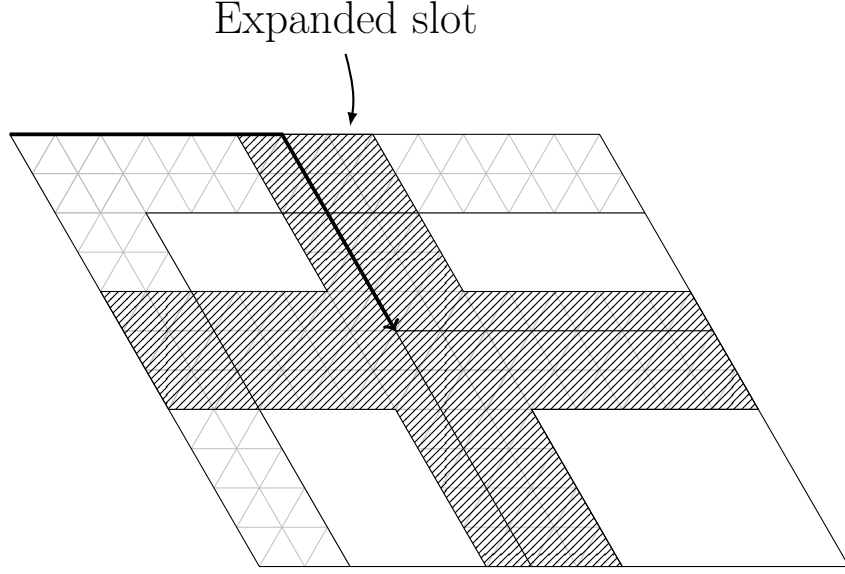


Figure 3: The First Extension of path P

and 5.

Now we are ready to describe how Bob should answer on Alice's requests.

1. Bob answers `ChooseAny(Green, Blue)`. If Alice chooses green, Bob extends P accordingly Figure 4, and Figure 3 otherwise.
2. Bob answers `ChooseAny(Green, Red)` and extends P as showed on Figure 4 regardless Alice's choice.
3. Bob answers `ChooseAny(Blue, Red)` and extends P as showed on Figure 3 regardless Alice's choice.
4. Bob answers `ChooseAny(Green, Blue)` and colors all uncolored vertices adjacent to requested vertex by red.

Note that all of these assignments do not violate the invariants. Path P is extended by empty vertices or colored by green and all adjacent vertices lies below P are empty or colored by blue. The fourth case guarantees satisfying Invariant 7, and 4 and 5 in the future.

At the start of the game the number of clear groups of segments in both directions is $\Omega(n)$. In a turn Alice can spoil at most two groups in each direction. Every time Alice spoils a group, Bob either gives `ChooseAny` answer or finishes the game. The latter means that Bob can not hold all invariants.

Every time Bob moves the buffer he skips only spoiled groups. So at the end of the game all slots in at least one direction will be spoiled. The latter guarantees that Bob gives $\Omega(n)$ `ChooseAny` answers.

Remind that Bob's strategy guarantees that a trichromatic triangle can not be found until all invariants are hold. So for any strategy of Alice Bob gives $\Omega(n)$ `ChooseAny` answers.

□

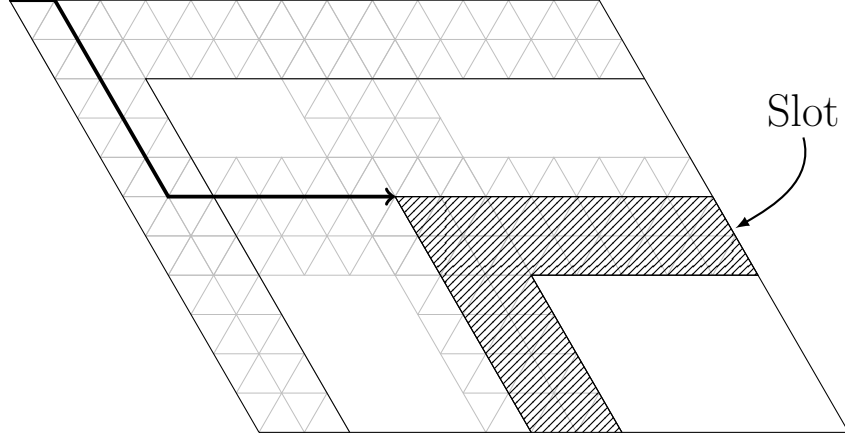


Figure 4: The Second Extension of path P

3.2 Arrows

We consider one more CSP, denote it as ψ_n , that corresponds to the game on a $n \times n$ square.

The variables correspond to the cells of the square and can take values from set $W = \{\leftarrow, \nearrow, \uparrow, \searrow, \rightarrow, \swarrow, \downarrow, \nwarrow\}$.

There are two types of constraints. 1) Two arrows in two cells with a common edge differ by at most 45° . 2) All arrows on the boundary of the $n \times n$ square should not be directed outside of the square. We set a constraint for each pair of adjacent cells and each cell on the border.

The unsatisfiability of ψ_n follows from Brouwer's theorem.

Theorem 3.1. CSP ψ_n is unsatisfiable.

Corollary 3.1. For the CSP ψ_n there exists a contradiction search tree of depth $O(n)$ and therefore of size $2^{O(n)}$.

For proofs of Theorem 3.1 and Corollary 3.1 see Appendix A.

Theorem 3.2. The size of any contradiction search tree for CSP ψ_n is $2^{\Omega(n)}$, and therefore the depth is $\Omega(n)$.

The proof of Theorem 3.2 follows from Lemma 3.1 and following one.

Lemma 3.4. For CSP ψ_n there exists such a strategy of Bob that Bob gives $\Omega(n)$ ChooseAny answers for any strategy of Alice.

Proof. Starting from the left side Bob mentally split the square on vertical strips $n \times 8$ that we will call *slots*. Since n may not be divisible by 8, there may be a strip of size at most $n \times 7$, which remains unsplit.

Let us divide a slot into two strips of size $n \times 4$. We say that left one is a buffer position. At the beginning of the game Bob divides the rightmost slot into two strips $n \times 4$ and chooses left one. We call that strip as a buffer. Bob fills all cells, which lie to the right of the buffer, by \leftarrow .

During the game Bob may move the buffer, but the new position for the buffer is determined in the same way. Bob divides a slot in two parts and chooses left one.

Bob maintains the following invariants: 1) The buffer is empty. 2) The area, to the right of the buffer, is filled completely. The leftmost strip $n \times 1$ to the right of the buffer is filled with \leftarrow . 3) All neighbours of empty cells are either empty or filled by \rightarrow . 4) Let k be the number of slots to the right of the buffer. Then Bob gave at least $\frac{k}{2}$ answers ChooseAny for cells to the right of the buffer. 5) No constraints are falsified.

Note that all invariants are hold at the begining.

The Bob's strategy has the following type: 1) If Alice requests already filled cell, then Bob returns its value. 2) If Alice requests empty cell, then Bob moves ChooseAny and probably fill some cells in order to maintain invariant. 3) If it is impossible to maintain invariants, Bob moves in arbitrary way.

We specify the second step and prove that if Bob fails to maintain invariants, then he has already made $\Omega(n)$ ChooseAny answers.

Let Alice request an empty cell c . We consider three cases: 1) the cell c is to the left of the buffer and does not share edges with cells from the buffer; 2) the cell c is in the buffer; 3) the cell c is in the $n \times 1$ strip that is adjacent to and to the left of the buffer.

In the first case Bob answers ChooseAny(\rightarrow , \searrow) if c is on the upper boundary and ChooseAny(\rightarrow , \nearrow), otherwise.

Bob assigns \rightarrow to empty neighbors of c . It is easy to see that such assignments do not violate invariants.

In the second case c lies in the buffer. We consider two cases a) there are no empty slots left of the buffer; b) there is an empty slot left of the buffer.

a) Assume that there is no empty slot left of the buffer. Every of t slots left of the buffer has a filled cell. Since for any request to an empty cell left of the buffer Bob returns ChooseAny and assigns values for no more than two slots, Bob has already made at least $\frac{t}{2}$ ChooseAny answers. Also Bob has made $\frac{k}{2}$ ChooseAny answers for cells to the right of the buffer, where k is the number of slots to the right of the buffer. Finally, Bob has made $\frac{t+k}{2} = \Omega(n)$ ChooseAny answers.

b) If there is an empty slot left of the buffer, Bob chooses the rightmost one and denote it by S . He divides the slot into two strips $n \times 4$ and chooses left one for new buffer B . We denote the old buffer by B' . We denote the area between S and B' by M . Bob is looking for an empty horizontal strip H of height 8 in M , that is not adjacent to lower and upper boundary. Note that if there is no such a strip, then Bob has made $\Omega(n)$ answers ChooseAny for cells in M .

Bob assigns \rightarrow to all empty cells from $M \setminus H$ and fills the parts $S \setminus B$ and $M \cap H$ as it is shown in Figure 5.

We split the old buffer B' into three parts: above, on, and below strip H . Bob assigns cells from B' in two ways: with Pattern 1 or Pattern 2, shown on Figure 5. Note that these two patterns has the following properties: there is no cell in strip H , which is filled with the same arrow in both patterns. So Bob can answer ChooseAny suggesting two variants according Patterns 1 or 2.

It is easy to see that all invariants are hold and constraints are not violated after such assignments. Let k be the number of slots that are to the right of buffer B' and s is the number of slots between old and new buffers. There are $k + s + 1$ slots to the right of buffer B . Bob made at least $\frac{k}{2}$ ChooseAny answers in the cells that are to the right of

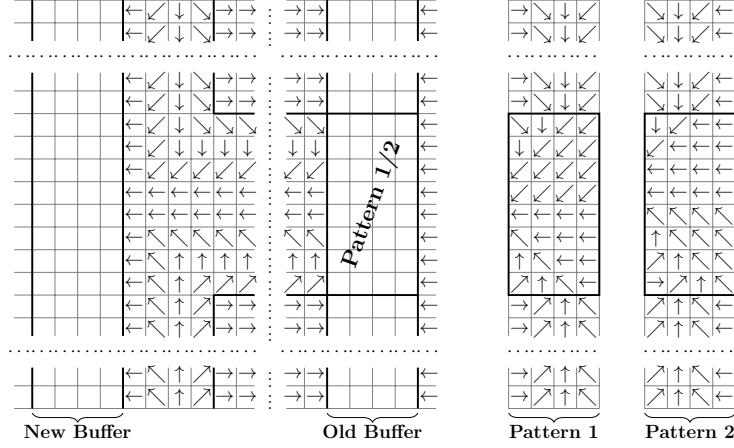


Figure 5: Assignments between buffers, patterns for old buffer

buffer B' , all s slots between B and B' are nonempty. The first request to every of those slots got the answer `ChooseAny` and after recovering of the invariants at most two slots may be affected. Thus we have at least $\frac{s}{2}$ answers `ChooseAny` for these s slots. Finally, the number of `ChooseAny` answers to the right of buffer B' is at least $\frac{s}{2} + \frac{k}{2} + 1 > \frac{s+k+1}{2}$.

In the third case c is a left neighbour of the buffer. In this case Bob gives answers `ChooseAny`(\rightarrow , \searrow) as it was in the first case. Bob has to fill the neighbours of cell c by \rightarrow , but some of them lie in the buffer. Thus we move the buffer as in the second case, but fill it with Pattern 1 (see Figure 5). So we do not violate the constraints. The number of `ChooseAny` answers are estimated as follows: $\frac{k}{2}$ answers to the right of B' and at least $\frac{s+1}{2}$ others (in contrast to the second case, there is no `ChooseAny` in B' but Bob gave the `ChooseAny` answer in at least half of $s + 1$ remaining slots). Thus there are at least $\frac{s+k+1}{2}$ `ChooseAny` answers to the right of the new buffer B . \square

4 Arrows is PPAD-complete

In this section we revisit the problems from the previous one, but with another computational model. In the first step we will give basic definitions of computational classes **FNP**, **TFNP** and **PPAD**. It is already known that **TFNP**-problem, based on Sperner's lemma, is **PPAD**-complete [3]. This section is devoted to **TFNP**-problem, based on arrows' game. We denote it as **ARROWS** and give the strict definition below. The main result, presented in this section, that **ARROWS** is **PPAD**-complete.

Definition 4.1 (FNP and TFNP). Let $R \subset \Sigma^* \times \Sigma^*$ be a polynomial-time computable relation such that there exists a polynomial p that for every $(x, y) \in R$ we have $y \leq p(|x|)$. The **NP** search problem Q_R specified by R is to given input $x \in \Sigma^*$ find $y \in \Sigma^*$ such that $(x, y) \in R$, if such y exists, or return 'no' otherwise. We use **FNP** to denote the class of **NP** search problems. An **NP** search problem is said to be total if for every x , there exists an y such that $(x, y) \in R$. We use **TFNP** to denote the class of total **NP** search problems.

Definition 4.2 (Polynomial Reduction). A search problem $Q_{R_1} \in \mathbf{TFNP}$ is polynomial-time reducible to a search problem $Q_{R_2} \in \mathbf{TFNP}$ if there exists a pair of polynomial-time

computable functions (f, g) such that for every input x in Q_{R_1} , if y satisfies $(f(x), y) \in R_2$, then $(x, g(y)) \in R_1$.

Definition 4.3 (LEAFD). The input of the problem is a pair $(M, 0^k)$ where M is the description of a polynomial-time Turing machine which satisfies

1. for every $v \in \{0, 1\}^k$, $M(v)$ is an ordered pair (u_1, u_2) where $u_1, u_2 \in \{0, 1\}^k \cup \{no\}$;
2. $M(0^k) = (no, 1^k)$ and the first component of $M(1^k)$ is 0^k .

M generates a directed graph $G = \langle V, E \rangle$ where $V = \{0, 1\}^k$ in the following way. An edge uv appears in E iff v is the second component of $M(u)$ and u is the first component of $M(v)$. The output is a directed leaf (in-degree + out-degree = 1) of graph G which is different from 0^k .

PPAD [6] is the set of total **NP** search problems that are polynomial-time reducible to **LEAFD**. By definition, **LEAFD** is complete for **PPAD**.

For every $n \geq 1$, let $B_n = \{\mathbf{p} = (p_1, p_2) \in \mathbb{Z}^2 \mid 0 \leq p_1 < n \text{ and } 0 \leq p_2 < n\}$.

The boundary of B_n is then the set of points $\mathbf{p} \in B_n$ with $p_i \in \{0, n-1\}$ for some $i \in \{1, 2\}$. For every $p \in \mathbb{Z}^2$, we define $K_p = \{q = (q_1, q_2) \in \mathbb{Z}^2 \mid q_1 \in [p_1, p_1 + 1] \text{ and } q_2 \in [p_2, p_2 + 1]\}$, the 2×2 square with left bottom corner at p .

An arrow assignment of B_n is a function f from B_n to $A = \{\uparrow, \nwarrow, \leftarrow, \swarrow, \downarrow, \searrow, \rightarrow, \nearrow\}$. It is said to be valid if for every $p \in B_n$ on the boundary B_n

1. if $p_2 = 0$, then $f(p) \in \{\nearrow, \uparrow, \nwarrow\}$;
2. if $p_2 = n-1$, then $f(p) \in \{\searrow, \downarrow, \swarrow\}$;
3. if $p_1 = 0$, then $f(p) \in \{\searrow, \rightarrow, \nearrow\}$;
4. if $p_1 = n-1$, then $f(p) \in \{\swarrow, \leftarrow, \nwarrow\}$.

Notice that for every corner there is only one valid arrow.

Definition 4.4 (ARROWS). The input of the problem **ARROWS** is a pair $(F, 0^k)$ where F is the description of a polynomial-time Turing machine which generates a valid arrow assignment f on B_{2^k} . Here $f(p) = F(p) \in A$ for every $p \in B_{2^k}$. The output is a pair of points $(\mathbf{p}, \mathbf{q}) \in B_{2^k}^2$ such that p and q are adjacent by edge and the angle between their arrows more than 45 degree.

In order to prove that **ARROWS** \in **PPAD** we use **2D - BROUWER** problem which is very similar to **ARROWS**. It is known that **2D - BROUWER** is **PPAD**-complete [3].

A 3-coloring of B_n is a function g from B_n to $\{0, 1, 2\}$. It is said to be valid if for every \mathbf{p} on the boundary of B_n ,

1. if $p_2 = 0$, then $g(p) = 2$;
2. if $p_2 \neq 0$ and $p_1 = 0$, then $g(p) = 0$;
3. otherwise, $g(p) = 1$.

The search problem **2D - BROUWER** is then defined as follows.

Definition 4.5 (2D – BROUWER, [3]). The input of the problem 2D – BROUWER is a pair $(F, 0^k)$ where F is the description of a polynomial-time Turing machine which generates a valid 3-coloring g on B_{2^k} . Here $g(p) = F(p) \in \{0, 1, 2\}$ for every $p \in B_{2^k}$. The output is a point $\mathbf{p} \in B_{2^k}$ such that K_p is trichromatic, that is, K_p has all the three colors.

Lemma 4.1. ARROWS \in PPAD.

Proof. We divide all arrows in three groups $\{\downarrow, \swarrow, \leftarrow\}$, $\{\rightarrow, \searrow\}$, $\{\nearrow, \uparrow, \nwarrow\}$ and assign each group value 0, 1, and 2, respectively.

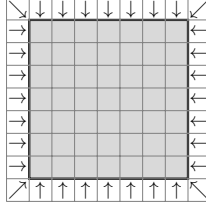


Figure 6: Square boundary

Now we surround the current square by arrows as it is shown on Figure 6, and convert new instance to 2D – BROUWER problem replacing arrows by values 0, 1, and 2 as they are divided on groups above.

Notice that built coloring is valid, and there should be a trichromatic square K_p , which will be an output of 2D – BROUWER problem. By construction square K_p can not touch the border of the big square.

On the other hand if square 2×2 is trichromatic then there should be two cells in it whose arrows directions differ on more than 45 degree. The latter can be easily checked since there is no arrow in one of the groups that is close up to 45 degrees to some arrows in two other groups. \square

To prove PPAD-hardness we use planar version of LEAFD problem.

For every $n \geq 1$ let us denote $V_n = \{u = (u_1, u_2) \mid 0 \leq u_1 < n \text{ and } 0 \leq u_2 < n\}$.

Definition 4.6 (RLEAFD). The input instance is a pair $(K, 0^k)$ where K is the description of a polynomial-time Turing machine which satisfies:

1. For every $u \in V_{2^k}$ $K(u)$ is an ordered pair (u_1, u_2) where $u_1, u_2 \in V_{2^k} \cup \{no\}$;
2. $K((0, 0)) = (no, (1, 0))$ and the first component of $K((1, 0))$ is $(0, 0)$.

K generates a directed graph $G = (V_{2^k}, E)$ on the grid in the following way. (u, v) is an edge iff v is the second component of $K(u)$, u is the first component of $K(v)$ and $|u_1 - v_1| + |u_2 - v_2| = 1$.

The output is a directed leaf (with in-degree + out-degree = 1) of graph G which is different from the origin $(0, 0)$.

Chen and Deng proved that RLEAFD problem is PPAD-complete [3].

Lemma 4.2. ARROWS is PPAD-hard.

Proof. We build a polynomial reduction from **RLEAFD** to **ARROWS** using the following gadgets. Let $G = \langle V_{2^k}, E \rangle$ be a graph from the definition of **RLEAFD**. We build the following grid as it is shown on Figure 7. Empty rectangle on the figure contains $2^k \times 2^k$ blocks, where every block is a 9×9 square. We fill block (i, j) according the information about vertex (i, j) in graph G and edges that are incident to (i, j) (see Figure 8).

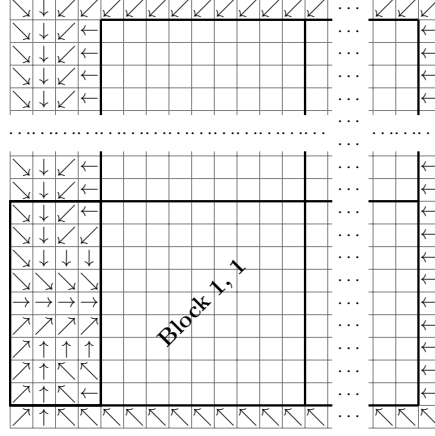


Figure 7: Border gadget

If the degree of the node is zero, then each cell in the block contains \leftarrow . Otherwise we build a path of \rightarrow that goes through the center of the block in direction of the corresponding path in graph G (see Figure 9).

It is easy to see that there are no any conflicts on the borders of the blocks and with the border gadget. So the only places with conflicts are ends of the paths in graph G . A value in any cell can be computed in polynomial time using a description of a polynomial-time Turing Machine from the instance of **RLEAFD**; it is also easy to compute in polynomial time a leaf in **RLEAFD** from the coordinates of the conflict in **ARROWS**. \square

The proof of the next theorem follows from Lemma 4.1 and 4.2.

Theorem 4.1. **ARROWS** is **PPAD**-complete.

Acknowledgements

The authors are grateful to Alexander Shen for fruitfull discussions and the statement of the problem and also thank Mikhail Slabodkin for helpfull comments.

References

- [1] Andrew B. Baker. Intelligent backtracking on constraint satisfaction problems: Experimental and theoretical results, 1995.
- [2] Maria Luisa Bonet, Juan Luis Esteban, Nicola Galesi, and Jan Johannsen. On the relative complexity of resolution refinements and cutting planes proof systems. *SIAM J. Comput.*, 30(5):1462–1484, May 2000.

- [3] Xi Chen and Xiaotie Deng. On the complexity of 2d discrete fixed point problem. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(037), 2006.
- [4] P. Crescenzi and R. Silvestri. Sperner's lemma and robust machines. *Computational Complexity*, 7(2):163–173, May 1998.
- [5] Cho Yee Joey Hwang. A Theoretical Comparison of Resolution Proof Systems for CSP Algorithms. Master's thesis, Simon Fraser University, 2004.
- [6] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, June 1994.
- [7] E. Sperner. Fifty years of further development of a combinatorial lemma. *W Forster (Ed.), Numerical Solution of Highly Nonlinear Problems, North-Holland, Amsterdam, New York*, 8:183–217, 1980.
- [8] Alasdair Urquhart. The depth of resolution proofs. *Studia Logica*, 99(1-3):249–364, 2011.

A Proofs from Section 3.2

The unsatisfiability of ψ_n follows from Brower's theorem, but we give an alternative proof that will be used in proving upper bound on the size of a contradiction search tree.

Let us consider arrows in two adjacent by edge cells and suppose the first arrow should be rotate on x degree in the clockwise way to get the second one. We also suppose that x is minimal in absolute value angle. The rotation between two arrows is signed x .

Now let us consider a closed path on the board, that goes through cells with common edges. We go along the path and calculate the sum of rotations between neighbouring arrows. Since the path returns back to the initial cell, the total rotation is divisible by 360° .

Lemma A.1. Consider a rectangle, with arrows in its cells, such that the total rotation on the closed path going along the boundary of the rectangle is nonzero. Then there are two cells with the common edge such that the angle between arrows in them is more than 45° .

Proof. We give the proof by induction on the size of rectangle. The base of the induction is 2×2 rectangle. Consider rectangle R with total rotation along the boundary S . We divide rectangle R on two parts as it is shown on Figure 10. Let S_1 and S_2 be total rotation along the boundary of smaller rectangles R_1 and R_2 . Note that $S_1 + S_2 = S$. Hence for one of the smaller rectangle the total rotation is nonzero. □

Theorem A.1. CSP formula ψ_n is unsatisfiable.

Proof. We prove that the total rotation along the boundary of $n \times n$ square is 360° . Consider upper left cell A and lower right cell B . Let the total rotation along the upper path from A to B is $l + 360^\circ k$, where $0 \leq l < 360^\circ, k \in \mathbb{Z}$. An arrow in the cell A belongs to set $\{\rightarrow, \searrow, \downarrow\}$ and one in the cell B belongs to set $\{\leftarrow, \swarrow, \uparrow\}$, therefore $l \neq 0$. Note that if $k \neq 0$, then on the path from A to B there exists \nearrow that contradicts to the boundary constraints. So we have that the total rotation from A to B is positive and less than 360° . Similarly the total rotation along lower path from B to A is positive and less than 360° . Thus the total rotation along the closed path is positive and less than 720° . But the total rotation should be divisible by 360° , therefore it equals 360° .

The Theorem follows from Lemma A.1. □

Corollary A.1. For the CSP formula ψ_n there exists a contradiction search tree of depth $O(n)$ and therefore of size $2^{O(n)}$.

Proof. Make a request to all boundary cells. After it we split a square into two approximate equal parts by a column. Since the total rotation on the boundary of two parts is nonzero, then by Lemma A.1 one of these parts has nonzero rotation and therefore contains a contradiction. We split the contradictory part by a row and reduce the problem of a contradiction search to a rectangle of size at most $(\frac{n}{2} + 1) \times (\frac{n}{2} + 1)$ with known arrows on the boundary. So we make approximately $1.5n$ requests and reduce the problem to two times smaller problem. The depth of the resulting tree is $O(n)$. □

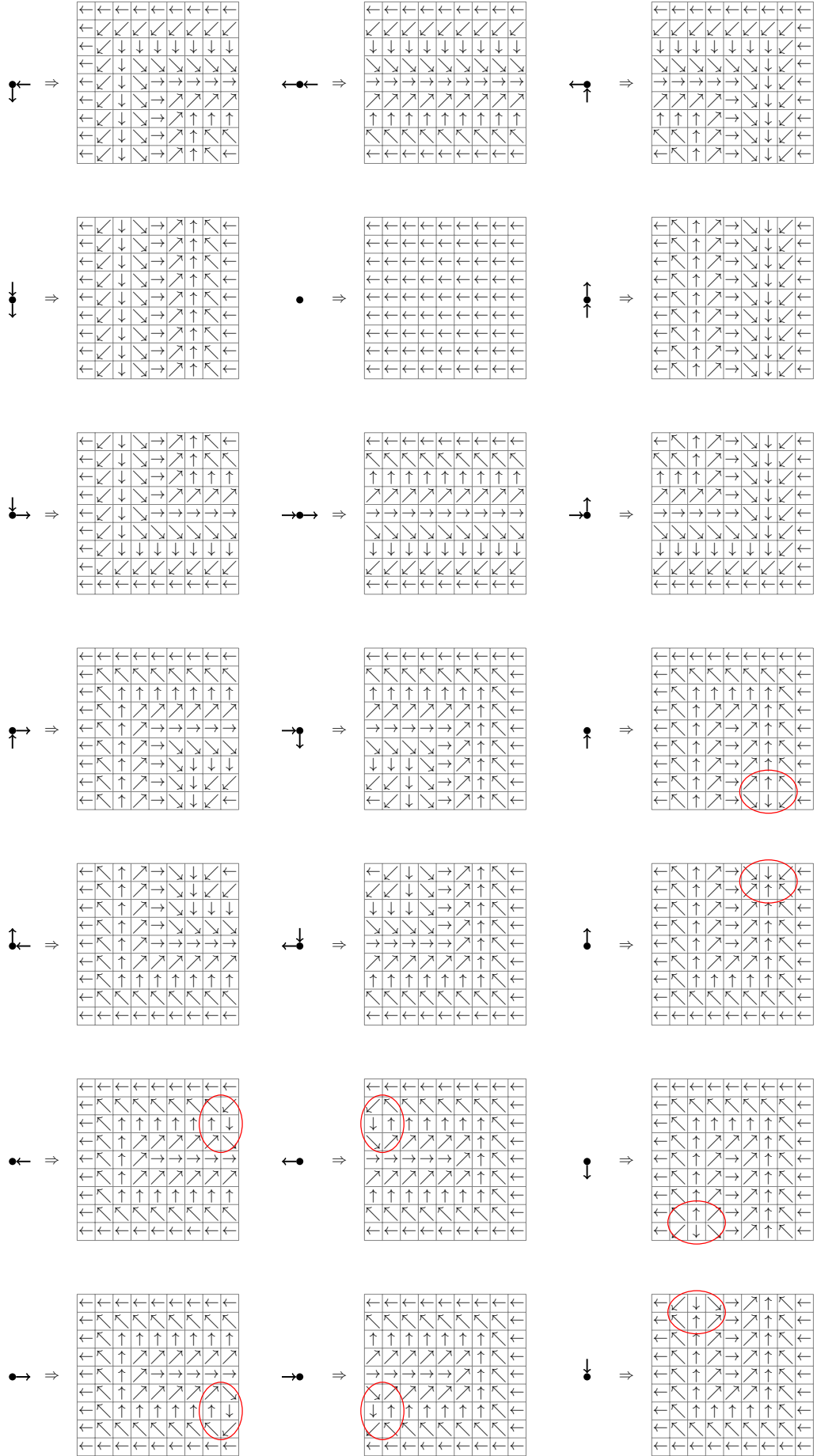


Figure 8: Block gadget

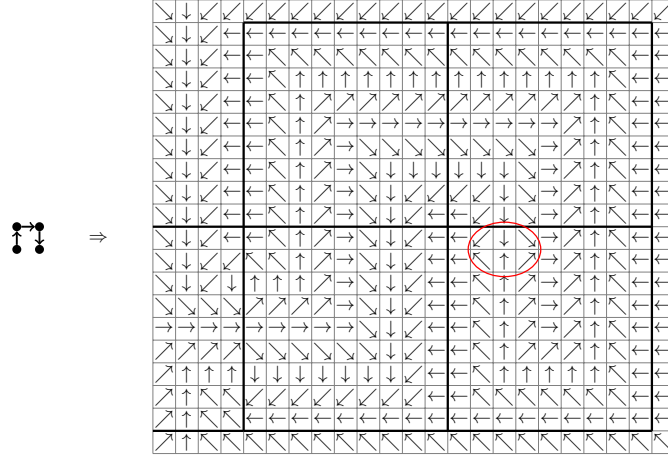


Figure 9: Example

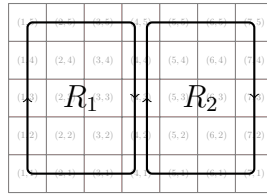


Figure 10: Division of R into R_1 and R_2 .